Руслан Богатырев

Природа и эволюция сценарных языков

Источник: Мир ПК, #11/2001.

Сценарные языки, или языки скриптов (scripting languages), за последние годы сделали огромный шаг вперед. Еще лет десять назад им отводилась роль вспомогательных средств, которые и называть-то языками программирования было как-то неловко. Сейчас же скепсис по отношению к ним сменился интересом и признанием. Но какова их природа и какое место они занимают среди всех языков?

Лисп как предтеча сценарных языков

Характеризуя ситуацию с вавилонским столпотворением языков программирования, Джон Бэкус, автор Фортрана и формы описания синтаксиса BNF (форма Бэкуса—Наура), писал (1977): «По-видимому, с языками программирования происходит что-то неладное. Всякий новый язык включает с небольшими изменениями все свойства своих предшественников плюс кое-что еще. Руководства по некоторым языкам занимают более 500 страниц...

В течение двадцати лет языки программирования неизменно развивались в одном и том же направлении, пока не дошли до нынешнего состояния «ожирения»... Теперь это излюбленная область тех, кто предпочитает возиться с пухлыми перечнями подробностей вместо того, чтобы бороться за новые идеи. Дискуссии о языках программирования часто напоминают средневековые диспуты о числе ангелов, которые могут разместиться на кончике иглы, а не волнующие споры о фундаментально различающихся понятиях... Достойно удивления, почему столь многие из нас, изучив отвратительные структуры типов традиционных языков с помощью изящного инструментария, разработанного Д. Скоттом, пассивно сохраняют верность этим структурам...»

В 1952 г. в швейцарском Базеле была издана работа Х. Рутисхаузера «Автоматическая разработка плана с помощью программно-управляемых вычислительных машин» (Automatische Rechenplanfertigung bei program-gesteuerten Rechenmaschinen, Birkhauser Verlag), с которой и ведет отсчет история изучения языков программирования. За прошедшие полвека появились тысячи языков, различных по своей природе и охватывающих разные модели и парадигмы программирования, включая программирование процедурное, функциональное, символьное, логическое, продукционное, реляционное, параллельное, объектно-ориентированное, модульное (компонентное) и др. Но по сути можно выделить два четких полюса притяжения языков: императивный и декларативный. Именно они и задают основу деления языков. Императивные языки скорее отвечают на вопрос «как?», тогда как декларативные — на вопрос «что?».

При этом императивные и родственные им языки обычно носят статический характер, в то время как декларативные имеют динамический.

В 1960 г. возникла идея увеличения общности, заключающаяся в том, чтобы воспользоваться логикой для такого описания фактов, которое не зависело бы от того, как эти факты будут использоваться впоследствии. Джон Маккарти, автор языка Лисп, вспоминает [1]: «Тогда мне казалось (как, впрочем, и сейчас), что люди по объективным причинам предпочитают общаться с помощью декларативных предложений, а не языков программирования, все равно, является ли субъект общения человеком, существом с Альфы Центавра или компьютерной программой... Основным преимуществом декларативной информации является ее общность».

Дж. Бэкус (IBM) и Дж. Маккарти (Массачусетский технологический институт), создавшие знаменитые языки Фортран (1954, Fortran) и Лисп (1958, Lisp) соответственно, задали точку отсчета эволюции языков программирования. Но если традиционные языки, пошедшие по стопам императивного Фортрана, известны достаточно хорошо и применяются весьма интенсивно, то языки другого направления с течением времени стали уходить в тень. Дж. Саммит, известный специалист по языкам программирования, как-то заметила, что все языки программирования грубо можно разбить на два класса. В одном находится Лисп, а в другом — все остальные. С этим трудно не согласиться, ведь Лисп и тогда, и сейчас заметно отличается от традиционных языков своими синтаксисом и семантикой, природой и реализацией.

Бестиповый язык Лисп задумывался Дж. Маккарти как средство для рекурсивных построений. В его основе лежит строгий математический аппарат лямбда-исчисления Чёрча, алгебра списочных структур (S-списки) и теория рекурсивных функций. Процедуры в нем могут служить в качестве данных, подставляемых на место других аргументов. В результате каждого действия возникает некое значение. Значения становятся аргументами следующих действий и т. д. Композиция и рекурсия — основные средства функционального программирования. Необычный синтаксис (скобочная префиксная запись) стал одной из причин сравнительно редкого использования Лиспа и его потомков в наши дни. В этом языке впервые нашли свое воплощение идеи интерпретатора, сборщика мусора, перегрузки операций и других хорошо сейчас известных механизмов. Числовая и символьная обработки стали одними из первых классов задач, для которых создавались языки программирования. Но если Фортран ориентируется на работу с числами, Си — на работу с символами и указателями, то Лисп — на работу с программами. Его структуры данных особенно полезны для представления и манипулирования исходными текстами программ. Это дает огромный потенциал для развития идей, но нередко затрудняет понимание сути для людей непосвященных.

Менее известен тот факт, что Лисп создал почву (идейную и операционную) для появления объектно-ориентированного программирования (ООП), ставшего едва ли не обязательной парадигмой для современных сценарных языков. Принято считать, что истоки ООП лежат в языках Симула (Кристен Нигаард, Оле-Йохан Даль, 1966) и Smalltalk (Алан Кей, 1971). Однако крестный отец языка Паскаль, известный английский ученый Тони Хоар, отмечая сильное влияние Лиспа на свои идеи концепции атрибутивных классов (record classes), уже в 1966 г. достаточно четко изложил суть ООП. Он писал [2]: «Фундаментальная особенность нашего понимания мира заключается в том, что мы систематизируем свой жизненный опыт, представляя его в виде отдельных понятий или объектов (столы и стулья, банковские займы и алгебраические выражения, многочлены и люди, транзисторы и треугольники и т. п.), и наше мышление, язык и действия основываются на обозначении, описании и манипуляциях с этими объектами, с каждым в отдельности или в связи с другими объектами... Всякий объект, представленный в запоминающем устройстве вычислительной машины в виде записи (record), будет обладать одним или более атрибутами (attributes), с которыми и приходится иметь дело при решении задачи... Объекты реального мира часто удобно классифицировать с помощью определенного числа классов (classes), причем любой класс обозначается некоторым собирательным именем, таким как «человек», «банковский заем», «выражение» и т. д.

Лисп отличается не только возможностями унификации представления данных в виде списков, создания основ работы с классами и объектами, а также оперирования с функциями в рамках лямбда-исчисления, но и наличием мощного аппарата макропрограммирования. По сравнению с Лиспом препроцессор языка Си крайне ограничен: его макроязык состоит только из подстановок и конкатенации символов. В нем нет ни рекурсии, ни метарекурсии, другими словами, макросы в Си не могут ни вызывать себя, ни определять другие макросы. Лисп же сам является метаязыком и способен решать сложнейшие задачи трансформации текста.

Недостатки сценарных языков. Заблуждения и стереотипы

Итак, Лисп, по всей видимости, был праотцем сценарных языков. Но что же такое сценарные языки? Это, пожалуй, ключевой вопрос, ответить на который отнюдь не просто. В отношении сценарных языков уже сформировались ложные стереотипы. В частности, это касается таких критериев оценки, как компиляция/интерпретация кода, система типов, быстродействие, требования к памяти, надежность. Склонность к поддержке интерпретации, а не компиляции кода считается едва ли не первым признаком сценарных языков. Интерпретаторы проще в исполнении, нежели компиляторы, и к тому же покрывают более широкий спектр языков. Однако те же традиционные языки Лисп, Снобол, Пролог, Форт и даже «пограничный» Бейсик чаще всего имеют реализации в виде интерпретаторов. Тогда как среди сценарных языков, хоть и нечасто, но можно встретить компиляторы. Например, на платформе Microsoft .NET реализованы компиляторы Perl и Python, порождающие промежуточный MSIL-код, исполняющая (с динамической компиляцией) в рамках среды Common Language Runtime. В настоящее время все чаще используют смешанные схемы, когда код частично компилируется, частично интерпретируется (это свойственно сценарным языкам в индустрии компьютерных игр).

Принято считать, что сценарные языки либо имеют слабую типизацию, либо вообще бестиповые. Это справедливо для части языков, но далеко не для всех. Более того, можно привести примеры бестиповых языков, не являющихся сценарными, взять хотя бы тот же BCPL, прародитель языка Си.

Представляет интерес проведенное в университете Карлсруэ эмпирическое сравнение языков Си, Си++, Java, Perl, Python, Rexx и Tcl по быстродействию и требованиям к памяти. Как отмечает Лутц Прехельт [3], «между средним временем выполнения программ на Java и сценариев нет существенной разницы. С вероятностью 80% сценарий будет выполняться в 1,29 раза дольше, а программа на Java по меньшей мере в 1,22 раза дольше, чем программа на Си или Си++». Что касается требований к оперативной памяти, то данные Прехельта тоже дают немалую пищу для размышлений: «Типичный сценарий занимает примерно вдвое больше памяти, чем программа на Си или Си++. Программы на Java занимают в три-четыре раза больше памяти, чем программы на Си и Си++».

Недостаточная надежность сценарных языков — тоже из разряда заблуждений. Так, яркий представитель сценарных языков 1990-х гг. — язык Python обладает средствами обработки исключений, построенными по образу и подобию аналогичного механизма в языке Modula-3. А ведь именно из него были заимствованы решения структурной обработки исключений (SEH), внедренные корпорацией Microsoft сначала в Си и Си++, а затем и в среду CLR (Common Language Runtime) платформы .NET.

Главная характерная черта для сценарных языков — динамическая природа, нередко позволяющая трактовать данные как программный код (и наоборот), а также простота освоения тех средств, которые буквально тут же дают видимый результат. Но это поверхностное наблюдение. Чтобы глубже разобраться в сути, необходимо выяснить, откуда пошли сценарные языки, для каких целей их создавали и что послужило катализаторами их развития.

Таблица 1. Классификация языков, предложенная Дж. Бэкусом

Характеристика	A. Простые операционные модели	B. Аппликативные модели	C. Модели фон Неймана
Основы модели	Точные и полезные	Точные и полезные	Сложные, громоздкие, бесполезные
Историческая чувствительность	Обладают памятью и исторической чувствительностью	Нет памяти, нет исторической чувствительности	Обладают памятью, исторически чувствительны
Семантика	Смена состояний с очень простыми состояниями	Редукционная семантика, нет состояний	Смена состояний со сложными состояниями
Ясность программ	Неясные и концептуально бесполезные	Ясные и функционально полезные	Умеренно ясные и не очень полезные концептуально

Классификация языков

Существуют разные подходы к классификации языков программирования. Все они в той или иной мере упрощают реальную картину и охватывают лишь отдельные характеристики языков. Сложность классификации понятна: 50 лет эволюции языков программирования привели к тому, что взаимопроникновение концепций языков, которые используют различные модели и парадигмы, достигло едва ли не своего апогея. Почти каждый новый язык представляет собой «гремучую смесь» разных концепций и механизмов. Одной из наиболее примечательных является классификация моделей языков, предложенная Дж. Бэкусом в 1977 г. В соответствии с ней выделяются три категории языков (табл. 1):

- Простые операционные модели (языки, основанные на конечных автоматах, машине Тьюринга);
- Аппликативные модели (языки на основе лямбда-исчисления Чёрча, системы комбинаторов Карри, чистого Лиспа);
- Модели фон Неймана (традиционные языки программирования).

С точки зрения такой классификации сценарные языки ближе всего к категории B. Если составить несколько таблиц, куда будут сгруппированы наиболее значимые и известные языки, которые по тем или иным причинам можно назвать сценарными, то получится четыре класса таких языков:

- командно-сценарные;
- прикладные сценарные;
- языки разметки;
- универсальные сценарные.

Аналогичная классификация для традиционных языков, включая указание поддерживаемых ими парадигм и наличие соответствующих международных стандартов, была представлена в работе [4].

Таблица 2. Командно-сценарные языки.

Название	Год появления	Разработчик языка	Организация, где был создан язык
Pilot	1962	-	IBM
JCL (Job Control Language)	1964	-	IBM
RPG	1965	-	IBM
MUMPS	1969	Окто Барнетт+	Massachusetts General Hospital
sh	1971	Стив Бурн	AT&T Bell Labs
Awk	1977	Альфред Ахо+	AT&T Bell Labs
csh	1978	-	UC Berkeley
Rexx	1979	Майкл Коулишоу	IBM UK Laboratories
AppleScript	1993	-	Apple Computer

Командно-сценарные языки

Командно-сценарные языки (табл. 2) зародились еще в 1960-х гг., когда возникла острая потребность в языках, обеспечивающих управление программами, иначе говоря, языках управления заданиями. Среди них наиболее известен JCL, разработанный для OS/360. Менее знаком Pilot, ставший, пожалуй, первой ласточкой среди сценарных языков. Он поддерживает всего два типа данных (строки и числа) и имеет крайне ограниченный набор команд (TYPE, ACCEPT, MATCH, JUMP, USE, COMPUTE, END, YES, NO).

Что касается JCL, то, по словам Фредерика Брукса [5], «если бы нужно было привести пример наихудшего из когда-либо созданных языков, это был бы OS/360 Job Control Language... Крупнейшей ошибкой было создание одного языка заданий для системы, предназначенной для работы многих различных языков программирования, вместо того, чтобы сделать по одному для каждого из языков... Однако когда мы переходили к высокоуровневым языкам программирования, то мы сделали JCL на уровне ассемблера... Я не был проектировщиком JCL. Но его делали под моим руководством... и я несу за него ответственность».

К командно-сценарному классу относятся многочисленные интерпретаторы команд CLI (command language interpreter), так называемые языки пакетной обработки (batch language) и языки для построения системных командных оболочек (яркий пример — sh, csh и их вариации для UNIX). Как правило, такие языки ориентируются не на интерактивный, а на пакетный режим обработки, когда участие человека на этапе выполнения сведено к нулю и все работает в непрерывном потоке. Эти языки не только могли непосредственно взаимодействовать с соответствующей операционной системой, но и снабжались средствами грамматического разбора программ и трансформации данных. С их помощью можно было создавать различные программные фильтры, используемые, в частности, в конвейере (pipe) системы UNIX. В число известных языков такого типа входят Awk, впервые появившийся в AT&T UNIX Version 7, а теперь ставший частью стандарта POSIX Command Language and Utilities. К ключевым особенностям языка, нашедшим впоследствии широкое применение в среде универсальных сценарных языков, можно отнести механизм регулярных выражений, без которого разбор текста производится не очень эффективно.

Языки этого класса ориентировались также на обработку системных событий, генерирование текста и высокоуровневый доступ к базам данных. Здесь стоит отметить язык RPG (Report Program Generator). Он до сего времени успешно применяется для создания отчетов из корпоративных БД, работающих преимущественно на мэйнфреймах (RPG/400 для компьютеров IBM AS/400).

Наиболее активно из языков этого класса в наши дни используется Rexx, созданный в исследовательских лабораториях IBM. По набору средств он мало чем отличается от универсальных сценарных языков, однако выполнен в виде классического блочно-структурированного процедурного языка и предназначен преимущественно для интеграции и расширения функциональности приложений.

Таблица 3. Прикладные сценарные языки.

Название	Год появления	Разработчик языка	Организация, где был создан язык
HyperTalk	1986	-	Apple Computer
Visual Basic	1990	-	Microsoft
JavaScript	1994	-	Netscape Communications, Microsoft
CorelScript	1995	-	Corel
LotusScript	1995	-	Lotus Development
VBScript	1995	-	Microsoft
Pnuts	2001	Тойоказу Томацу	Sun Microsystems

Прикладные сценарные языки

Прикладные сценарные языки (табл. 3) зарождались в 1980-е гг., в эпоху появления промышленных ПК, когда на первый план стали выходить задачи интерактивного общения с ОС, а также доступа к данным электронных таблиц и БД. Отличительная особенность сценарных языков данного класса — ориентация на клиентскую часть ПО.

Использование объектной модели в языках данного класса уже становится нормой, а не исключением. Их еще нельзя назвать полноправными языками ООП, однако они в значительной мере стараются воспользоваться удобствами объектного подхода.

Среди прикладных сценарных языков резко выделяется Visual Basic, в том числе и такая его разновидность для офисного программирования, как VBA (Visual Basic for Applications). Visual Basic — это тот самый «пограничный» язык, который скорее относится к сценарным, чем к традиционным. Он во многом задал тон такому классу языков, как прикладные сценарные. Более того, работа с пользовательским интерфейсом и встраивание программных компонентов (VBX, OCX, ActiveX) стали едва ли не визитной карточкой данного языка. Стоит заметить, что в ходе эволюции VBA поглотил другие специфические языки, в частности Word Basic и Excel Macro Language, взяв на себя их задачи. Под его влиянием были созданы такие языки, как VBScript (особый диалект языка Visual Basic, ориентированный на создание OLE-компонентов и на работу в рамках браузеров) и LotusScript (своего рода аналог языков VBA и CorelScript, предназначенный для решения задач автоматизации офиса в рамках Lotus Notes).

Несколько особо в этом ряду стоит JavaScript, ставший стандартом де-факто в Web-программировании при реализации клиентской части. Его прототипом был язык LiveScript, являвшийся частью серверного продукта LiveWire компании Netscape и первоначально встроенный в Netscape Navigator 2.0. После появления языка Java корпорации Sun Microsystems он начал играть роль самодостаточной надстройки над этим языком, и его название сменилось на JavaScript. Диалекты этого языка — JScript корпорации Microsoft и ECMAScript (стандарт ECMA-262).

Из новичков в данном классе языков упомянем экспериментальный язык Pnuts, основная идея которого — дать в рамках сценариев наиболее полный доступ к Java API. Его можно использовать для самых разных задач, но прежде всего для оперирования компонентным ПО (подробнее см.: http://javacenter.sun.co.jp/pnuts).

Таблица 4. Языки разметки.

Название	Год появления	Разработчик языка	Организация, где был создан язык
GML	1969	Чарльз Гольдфарб+	IBM
TeX	1979	Дональд Кнут	Stanford University
SGML	1986	-	ISO
HTML	1991	Тим Бернерс-Ли+	CERN
CFML (Cold Fusion)	1995	-	Allaire
DHTML	1996	-	Microsoft, Netscape Communications
XML	1997	-	W3C
XHTML	2001	-	W3C

Языки разметки

Языки разметки, или тегированные языки (табл. 4), стоят несколько поодаль от магистральной линии развития сценарных языков. Им ближе всего по своей природе системы макрообработки (всевозможные макропроцессоры), столь популярные в 1960—1970-е гг. Их главная отличительная черта — встраивание специального кода (в виде обособленных «команд» — тегов) непосредственно в обычные тексты. Им родственны такие языки, как Postscript и RTF (чаще воспринимаемый просто как особый формат представления документов). Теги стали использоваться для самых разных целей: для отделения структуры информации от ее содержания, для вкрапления команд форматирования и даже для задания динамического поведения встроенных в документ интерактивных объектов.

Идея отделения структуры информации от содержания возникла давно. Но, пожалуй, первым осознанным решением стал запуск проекта GenCode. В сентябре 1967 г. Уильям Танниклифф, председатель Комитета по композиционным решениям ассоциации GCA (Graphic Communications Association), предложил провести четкую границу между содержанием и форматом представления информации. Руководитель GCA Норман Шарпф принял решение приступить к реализации соответствующего проекта GenCode.

Под влиянием GenCode в 1969 г. американский ученый Чарльз Гольдфарб возглавил работу исследовательской группы в IBM, целью которой была проработка принципов интегрированных информационных систем в области законодательства. Плодом усилий этого коллектива, куда входили также Эдвард Мошер и Реймонд Лори, стал GML — обобщенный язык разметки (Generic Markup Language, Goldfarb-Mosher-Lorie). Многие решения этой группы нашли применение в различных издательских системах IBM. На основе GML и идей системы Scribe, разработанной Брайаном Рейдом, Международная организация по стандартизации (ISO) разработала метаязык SGML (стандарт ISO-8879:1986).

Наиболее значительными достижениями в области языков разметки стали TeX, HTML и XML. Язык TeX (1979) Дональда Кнута на три года опередил Postscript, созданный Джоном Уорноком и др. в компании Adobe и предназначенный для точного описания внешней формы документов с композицией произвольной сложности. В отличие от низкоуровневого Postscript, язык TeX ориентировался на работу пользователей, не имеющих навыков программирования. Наиболее широко этот язык стал применяться в научной среде, где предъявляются самые высокие требования к качеству построения формул сложной структуры.

Язык HTML, с появлением которого понятие «гипертекст» стало простым и обыденным, создавался на базе SGML путем максимального упрощения его структуры и свойств. Вряд ли имеет смысл описывать его подробно: это основной язык представления информации в Web-среде, включая Интернет.

Метаязык XML, созданный во многом с подачи Джона Босака (Sun Microsystems), руководителя рабочей группы SGML ERB, также основывался на SGML, но для него характерно куда более бережное отношение к идеям своего предка, да и выполняет он совсем иную роль, нежели HTML. Это своего рода язык транспортирования и промежуточного хранения данных при обмене ими между разнородными и распределенными системами. На его основе можно проводить сколь угодно сложные преобразования документов и текстовой информации, а главное, в унифицированном виде хранить данные реляционно-иерархической структуры, в том числе по настройкам и программированию компонентов. Это, правда, не мешает использовать его и не совсем по назначению — для задания динамики поведения всевозможных объектов. В 2001 г. появилась ревизия HTML, которая получила название XHTML, где были учтены требования XML.

CFML (Cold Fusion) и DHTML — языки разметки, напрямую предназначенные для динамического создания Web-страниц и доступа к БД. Оба они являются расширениями HTML, но CFML — чисто коммерческий язык компании Allaire, не имеющий других реализаций, а DHTML, возникший из Dynamic HTML, — результат многочисленных компромиссов и конкуренции Netscape и Microsoft, которые были подытожены консорциумом W3C. Близкие им ASP (Active Server Pages, Microsoft) и JSP (Java Server Pages, Sun) — уже не языки, а, скорее, технологические «плавильные печи». Тот же ASP по сути не что иное, как сценарий на VBScript, который исполняется на сервере.

Таблица 5. Универсальные сценарные языки.

Название	Год появления	Разработчик языка	Организация, где был создан язык
Perl	1986	Ларри Уолл	
Tcl	1990	Джон Устерхаут	UC Berkeley
Python	1991	Гвидо ван Россум	Stichting Mathematisch Centrum
Ruby	1993	Юкихиро Матсумото	
Euphoria	1993	Р. Крейг	Rapid Deployment Software
Lua	1994	У. Целес+	PUC-Rio
PHP	1995	Расмус Лердорф	
Mawl	1995	Д. Лэдд+	Lucent Bell Labs
Pike	1996	Фредерик Хьюбинетт	InformationsVavarna
Curl	2000	Стив Уард+	MIT Lab for Computer Science

Универсальные сценарные языки

Представители этого класса языков наиболее широко известны (табл. 5). Именно их чаще всего и ассоциируют с термином «сценарный язык», причем применительно к Web-среде. При этом нередко упускается из виду тот факт, что создавались самые популярные из них совсем не для Web-программирования: языки Perl, Tcl и Python появились еще тогда, когда не было даже первой версии HTML.

Все они вышли из операционной системы UNIX. Но для каких же целей их создавали? Язык Perl (Practical Extraction and Report Language), что и отражено в расшифровке его названия, выполнял функции, сходные с тем, на что ориентирован RPG: управление данными и генерирование отчетов. Такой известности и популярности Perl достиг во многом благодаря тому, что с появлением спецификации CGI (Common Gateway Interface) он стал главным поставщиком серверных скриптов для формирования Web-страниц по запросу. Он оказался в нужное время в нужном месте. К тому же в нем прямо под рукой оказались такие удобные средства, как ассоциативные массивы (хеш-структуры) и регулярные выражения, выполняемые для традиционных языков в виде вспомогательных библиотек. Именно регулярные выражения автор языка Ларри Уолл отметил как важнейшее достоинство Perl [6].

Язык Python задумывался для обеспечения удобного доступа к системным средствам экспериментальной операционной системы Amoeba. Его автор Гвидо ван Россум под влиянием синтаксиса языка ABC и механизмов языков Modula-2+ и Modula-3, разработанных в исследовательском центре Systems Research Center корпорации Digital, к 1991 г. создал объектно-ориентированный язык сценарного характера. Его особенностями стали механизм лямбда-вычислений, заимствованный из языков функционального программирования, и расширенная обработка строк. Интерпретатор Python позднее получил версию компилятора в Java-код (JPython), а затем и компилятор в MSIL-код для платформы Microsoft .NET.

Цель создания языка Tcl (Tool Command Language) была совсем иной — интенсивная обработка строк и тесная интеграция с пакетом Tk, обеспечивающим удобное построение пользовательского интерфейса для любых интерактивных приложений. Он позиционируется как язык расширения приложений (application extension language). Джон Устерхаут, автор языка Tcl и инструментария Tcl/Tk, создавал эти средства для решения конкретной задачи (проектирование СБИС), находясь под влиянием работ исследовательской лаборатории Western Research Lab корпорации Digital. В отличие от Perl и Python язык Tcl в значительной мере опирается на подключение расширений, написанных на языках Си и Си++. Язык Tcl за счет подключаемого модуля SafeTcl нашел свое применение и в Web-программировании для поддержки в рамках браузера специальных аплетов, названных тиклетами (Tclets).

Остальные универсальные сценарные языки, упомянутые в табл. 5, принадлежат ко второй волне, возникшей при появлении HTML и Web-серверов. Самым известным из них является PHP (Personal Home Pages). При его создании Расмус Лердорф опирался на идеи JavaScript, Си и Bourne shell — известной командной оболочки ОС UNIX. (О языке Curl подробнее см. в работе [7].) Подавляющее большинство универсальных сценарных языков используются для реализации серверной части Web-систем, причем поддержка классов и объектов в том или ином виде присутствует практически во всех этих языках.

Природа сценарных языков

Теперь, когда мы кратко познакомились с отдельными представителями семейства сценарных языков, проще ответить на поставленный ранее вопрос: что же такое сценарный язык? Воспользуемся некоторыми положениями, сформулированными «отцами-основателями» данного направления.

Дихотомия Устерхаута — принцип классификации языков, предложенный Джоном Устерхаутом [8], автором языка Tcl. В соответствии с его подходом высокоуровневые языки делятся на языки системного программирования и на сценарные. Последние, называемые также языками склейки (glue languages), либо слабо типизированные, либо вообще бестиповые. Они не имеют средств для представления сложных структур данных и обычно интерпретируются. Сценарии, как правило, взаимодействуют либо с другой программой (зачастую как склейка), либо с набором функций, предоставляемых интерпретатором. (Склейка (glue) — на компьютерном жаргоне обозначает любую интерфейсную логику или протокол для взаимодействия двух компонентных блоков.)

Таблица 6. Сценарий, фильм, программа.

Театр	Кино	Web-программирование
Режиссер	Режиссер-постановщик	Руководитель проекта
Автор пьесы	Автор сценария	Автор
Художник	Художник	Художник/дизайнер
Актер	Актер	-
-	Оператор	Дизайнер-технолог
-	Монтажер	Программист

Ларри Уолл, автор языка Perl, исповедует несколько иной подход [см. 6]. По его словам (1998), сценарные языки должны формировать «свободную от этики артистическую обстановку». «Сценарий, — подчеркивает он, — это то, что вы передаете актеру, а программа — это то, что передается зрительному залу».

Продолжая мысль Уолла, можно провести простую аналогию между созданием Web-программ, театральным спектаклем и съемкой кинофильма (табл. 6).

> Сценарий → спектакль. Сценарий в театре «интерпретируется» актерами. В итоге зрительный зал видит спектакль.
>
> Сценарий → кинофильм. Сценарий в кино «компилируется» в съемочный материал, который монтажер компонует в фильм. В итоге зрительская аудитория видит кинофильм.
>
> Сценарий → программа. Сценарий в Web-программировании автоматически (путем интерпретации/компиляции) превращается в программу. В итоге пользователь видит результат работы программы.

Аналогия эта близка такой отрасли, как индустрия развлечений, прежде всего это относится к играм и авторским программам (authoring). Сценарные языки всегда весьма активно используют в компьютерных играх, в частности, это касается языков SCUMM, INF, COG [9]. Правда, по большей части они остаются закрытыми, являясь ноу-хау. Для стратегических игр, action-игр и даже имитаторов сценарии практически незаменимы, поскольку позволяют перекладывать вопросы программирования сцен, стратегий и сюжетов на плечи дизайнеров. Применение сценарных языков (хотя и специфическая реализация) в области компьютерных игр, где крайне важны надежность, объем памяти и скорость выполнения кода, — лучшее свидетельство их потенциальной эффективности.

Как отмечает Роберт Хюбнер [9], ведущий программист компании LucasArts Entertainment, «внутренний сценарный язык позволяет создавать отдельную среду вокруг вашей сервисной машины (engine). Это защищает виртуальную машину, выполняющую сложный и часто изменяющийся игровой код, от «реальной» машины, управляющей игровой сервисной машиной... Так как система сценариев гораздо более гибкая, нежели совокупность фиксированных эффектов, ваша сервисная машина сможет выполнять более интересные функции, которые первоначально и не планировались».

Какие же требования предъявляются к сценарному языку? Он должен служить средством быстрого макетирования. Грань между сценарием (рассматриваемым как полуфабрикат программы) и законченным продуктом должна быть столь незаметной, что только сам автор смог бы понять, когда макет превратился в конечный продукт (в зависимости от соответствия начальным требованиям).

Сценарный язык должен ориентироваться на скорость и простоту освоения базовых возможностей, быстро дающих видимый результат. Но из этого не следует, что язык должен быть примитивным. Путь к решению обычных типовых задач даже для начинающего должен быть коротким, простым и понятным. В этом отношении он должен напоминать родной язык: освоить несколько слов и фраз бывает достаточно, чтобы тебя понимали, однако научиться четко, грамотно и красиво выражать мысли на родном языке не всем и не всегда удается даже к глубокой старости.

Сценарный язык должен в меньшей степени опираться на создание конечного продукта с нуля и в большей степени — на использование тех мощностей, которыми обладает операционная система, графическая среда, прикладная сервисная машина и прочие подобные компоненты, вокруг которых строится «обвязка» в виде сценариев. Он в первую очередь обеспечивает удобную работу на уровне текстовых строк, стараясь по возможности не прибегать к интенсивному использованию сложных типов данных.

Итак, в дополнение к привычным понятиям алгоритма и программы добавилось новое — сценарий. В этой связи логично делить языки на алгоритмические, программные и сценарные. Как программные и сценарные языки могут сосуществовать друг с другом в разработке ПО? По оценке CI Labs Meta Group (1994), от 70 до 90% функциональности нового приложения обеспечивается за счет готовых модулей, тогда как для настройки и привязки приложения требуется всего 10–30% дополнительного программирования. Напрашивается вполне очевидный вывод, что наиболее критичные модули и сервисные машины должны выполняться на программных языках, тогда как использование их возможностей и встраивание в реальную среду разумнее производить с помощью сценарных. Аргументом в защиту такой позиции может служить вывод, сделанный Л. Прехельтом [3] в ходе проведенного им исследования: «Проектирование и составление программ на языках Perl, Python, Rexx и Tcl занимает не более половины времени, необходимого для программирования на Си, Си++ и Java, а длина исходного текста вдвое меньше». Это лишний раз подтверждает правило Барри Боэма [10], согласно которому количество строк исходного теста, создаваемого программистом в единицу времени, не зависит от языка программирования.

Возможная ниша сценарных языков в рамках создания программных систем по отношению к традиционным программным языкам указана в работе [11]. Коротко говоря, суть сводится к следующему: любая программа строится на трех образующих – данных, логике и внешнем интерфейсе. Если для удобства взять за основу популярную концепцию MVC (Model/View/Controller), то данные, логика и интерфейс отобразятся соответственно на модель, контроллер и вид. Предполагается, что модель обычно выступает в одном экземпляре, видов может быть много, а контроллер играет роль связующего слоя между первым и вторым. По аналогии с концепцией MVC все модули программной системы при таком подходе могут быть разбиты на три вида:

- машины (системный слой, модель);
- преобразователи (связующий слой, контроллер);
- сценарии (прикладной слой, вид).

Под машиной может пониматься любая сервисная машина и приравненный к ней компонент. Преобразователи могут быть универсальными и специализированными. Универсальные имеют общий характер (привычные библиотечные модули); специализированные соединяют машины и сценарии. Их удобнее всего выполнять на языках ООП. Деление на три вида совсем не обязательно должно предусматривать соответствие машин, скажем, уровню модели. Машина может использоваться и на уровне контроллера, если это удобно для данной задачи.

Взаимосвязь между программными и сценарными языками можно проследить и на модели Ершова [12]. В соответствии с ней выделяются три взаимодополняющих друг друга вида программирования:

- синтезирующее (формирование программных фрагментов/компонентов);
- сборочное (сборка программы из готовых фрагментов/компонентов);
- конкретизирующее (адаптация многопараметрической программы к особым условиям ее применения).

Как нетрудно заметить, традиционные языки в значительной степени доминируют в синтезирующем программировании, тогда как сценарные — в конкретизирующем. Их соприкосновение происходит в сборочном программировании, которое становится пограничной зоной двух разных языковых миров. До сих пор остается непонятным, является ли программирование по своей природе индуктивным или дедуктивным процессом. Это еще одна точка различия программных и сценарных языков. Первые исповедуют скорее индуктивный подход, тогда как вторые — дедуктивный.

Как известно, многочисленные попытки создать единый всеобъемлющий язык программирования закончились неудачно. Но это касалось именно языка, а не языковой среды и не лингвистической системы. Здесь важно отметить, что А.П. Ершов был автором не менее интересной идеи — лексикона программирования, который он понимал как «лингвистическую систему с фразовой структурой, содержащую в себе формальную нотацию для выражения всех общезначимых конструкций, употребляемых при формулировании условий задачи, при синтезе и преобразовании программ». Сценарные языки на их нынешнем этапе развития, пожалуй, ближе всего к этой идее.

Что касается формы сценарных языков (их синтаксиса), то немалое влияние на нее оказал и продолжает оказывать язык Си. Заимствование содержательных идей сценарными языками ведется по широкому фронту:

- языки функционального программирования (Лисп, Scheme);
- языки обработки строк (Snobol, Icon);
- объектно-ориентированные языки (Smalltalk, Java, Eiffel, C#);
- языки управления заданиями, командные языки (csh, Rexx);
- языки управления средой (Tcl, VBA);
- языки разметки/макрообработки (SGML, TeX);
- языки моделирования дискретных систем (GPSS, SIMSCRIPT, SLAM II).

В чем же еще проявляется воздействие программных (традиционных) языков на проектирование и развитие языков сценарных? Здесь можно выделить три направления:

- использование ООП (практически любой новый сценарный язык поддерживает объектную модель);
- использование идей старых языков (Лисп, Снобол; языки функционального программирования и языки обработки строк по своей динамической природе наиболее близки сценарным);
- появление языка Java (ряд сценарных языков наиболее тесно интегрируется именно с Java).

Несмотря на огромные усилия, которые предпринимали лидеры программной индустрии во главе с Sun Microsystems, язык Java в области Web-программирования не сумел занять господствующих высот. Тогда как в области промежуточного ПО, создания компонентов и серверных решений его роль весьма заметна. Ниша, не занятая языком Java, оказалась востребованной, и это создало почву для появления таких языков, как Curl.

По какому же пути пойдет развитие сценарных языков, какие идеи будут служить катализаторами процесса дальнейшего их совершенствования? В определенном смысле ответы на эти вопросы содержатся в словах знаменитого Дж. Бэкуса (1977): «Имеются многочисленные свидетельства того, что функциональный стиль программирования может по своей мощи превзойти стиль фон Неймана. Поэтому для программистов представляет большое значение разработка нового класса исторически чувствительных моделей компьютерных систем, которые воплощают такой стиль и не страдают неэффективностью, по-видимому присущей системам, основанным на лямбда-исчислении. Только когда такие системы и их функциональные языки докажут свое превосходство над традиционными языками, мы получим экономическую основу для разработки нового вида компьютера, который сможет наилучшим образом реализовать их».

Литература

1. Маккарти Дж. Общность в системах искусственного интеллекта // Лекции лауреатов премии Тьюринга за первые двадцать лет. 1966—1985. М.: Мир, 1993.
2. Хоар Т. Обработка записей // Языки программирования// — Programming Languages NATO Advanced Study Institute, IBM.— Paris France, 1968. /Пер. с англ. М.: Мир, 1972.
3. Прехельт Л. Эмпирическое сравнение семи языков программирования // Открытые системы. 2000. № 12.

4. Богатырев Р. Летопись языков. Паскаль // Мир ПК. 2001. № 4.
5. Brooks F. P. Language Design as Design // History of Programming Languages. ACM Press, 1996.
6. Kim E. E. A Conversation with Larry Wall // Dr.Dobb's Journal. 1998. February..
7. Богатырев Р. Технология Curl и концепция X Internet // Мир ПК. 2001. № 9.
8. Ousterhout J. Scripting: Higher Level Programming for the 21st Century // IEEE Computer. 1998. March.
9. Huebner R. Using Scripting Language in Games // Game Developer, 1997.
10. Boehm B.W. Software Engineering Economic. Prentice-Hall, 1981.
11. Богатырев Р. Золотой треугольник // Мир ПК. 2001. № 6—7.
12. Ершов А.П. Избранные труды. Новосибирск: Наука, 1994.